Date=18/11/2020
Lecture By=Manish Mahant
Subject ⇒States And Props 2

IN PREVIOUS LECTURE **(QUICK RECAP) Date-17/11/2020**	**In Today's Lecture (Overview)**
React State Creating the state Object Example: Using the state Object React Props React Props Example Question For Self-Practice	Re-Render Component In React… The setState() method Code The forceUpdate() method Code How to use React Memo and what memoization actually means Questions For Self-Practice / Assignment For The Day

Re-Render Component In React…

React components automatically re-render whenever there is a change in their state or props. A simple update of the state, from anywhere in the code, causes all the User Interface (UI) elements to be re-rendered automatically.

However, there may be cases where the `render()` method depends on some other data. After the initial mounting of components, a re-render will occur when:

- A component's `setState()` method is called.
- A component's `forceUpdate()` method is called.

The `setState()` method

If our component has `state`, we could simply update the `state` to its current value:

```
ourMethod(){
    // It simply sets the state to its current value
    this.setState({ state: this.state });
};
```

Code

In the following example, the `setState()` method is called each time a character is entered into the text box. This causes re-rendering, which updates the text on the screen.

```
import React, { Component } from 'react';
import 'bootstrap/dist/css/bootstrap.css';

class Greeting extends Component {
  state = {
    fullname: '',
  }

  stateChange = (f) => {
    const {name, value} = f.target;
    this.setState({
      [name]: value,
    });
  }
```

```
  render() {
    return (
      <div className="text-center">
        <label htmlFor="fullname"> Full Name: </label>
        <input type="text" name="fullname" onChange={this.stateChange} />
        <div className="border border-primary py-3">
            <h4> Greetings, {this.state.fullname}!</h4>
        </div>
</div>
    );
  }
}

export default Greeting;
```

The `forceUpdate()` method

Calling `forceUpdate()` will cause `render()` to be called on the component and skip `shouldComponentUpdate()`.

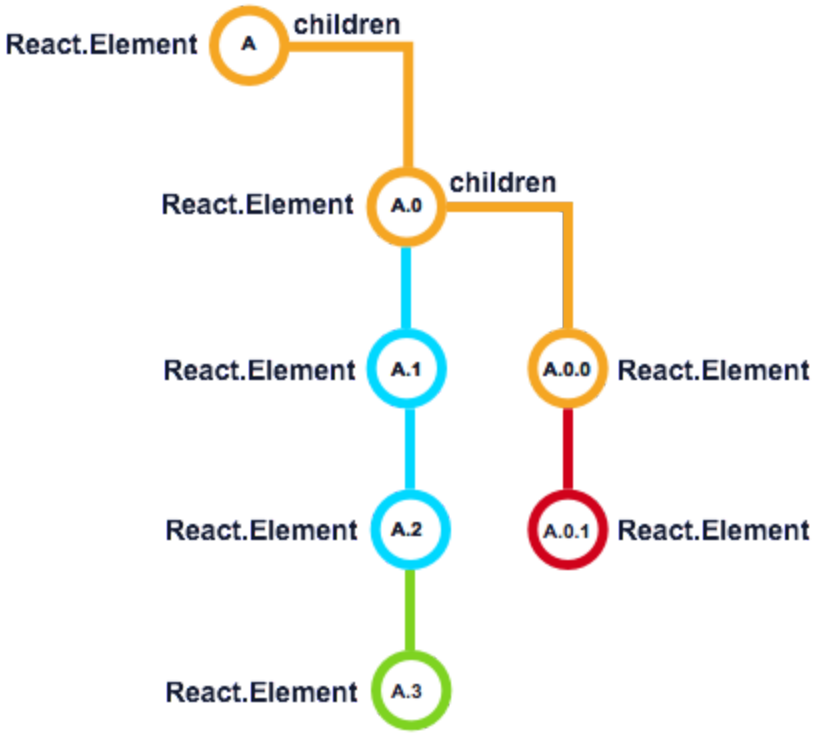
This will trigger the normal lifecycle methods for child components, including the `shouldComponentUpdate()` method of each child. React will still only update the DOM if the markup changes.

Code

The following example generates a random number whenever it loads. Upon clicking the button, the `forceUpdate()` function is called which causes a new, random number to be rendered:

```
import React, { Component } from 'react';
import 'bootstrap/dist/css/bootstrap.css';



class App extends React.Component{
  constructor(){
    super();
    this.forceUpdateHandler = this.forceUpdateHandler.bind(this);
```

```
  };

  forceUpdateHandler(){
    this.forceUpdate();
  };

  render(){
    return(
      <div>
        <button onClick= {this.forceUpdateHandler} >FORCE UPDATE</button>
        <h4>Random Number : { Math.random() }</h4>
      </div>
    );
  }
}

export default App;
```



How to use React Memo and what memoization actually means

React came out with some new goodies in version 16.6. One of ‘em is memo. A higher order component that can be used as a performance optimiser.

Memo → memoization?

The memo part in React.memo is a derivative from memoization. If you don’t have a hardcore computer science background, like me, it may throw you off a little bit. Let’s see what Wikipedia can tell us about memoization:

The most important thing we learn from this is that not even Wikipedia knows how to spell memoization.
Second, it tells us that it’s a technique that executes a (pure) function once, saves the result in memory, and if we try to execute that function again with the same arguments as before, it just returns that previously saved result without executing the function again.
And that kinda makes sense, doesn’t it? If the arguments are the same as last time, the outcome will be the same as well. So no need to execute that whole damn piece of code again.
The memoization in React.memo
If we pull that definition in to our React ecosystem, the functions that we were talking about, are our React components. And the arguments that we talked about, are our props.
Why do we need React.memo?
Just for the sake of an example, let’s create a container component that uses setState every second.

```
class App extends React.Component {
    names = ["Peter", "Bruce", "Clark"];
    state = { name: "Anonymous" };

    componentDidMount() {
      setInterval(() => {
        const name = this.generateName();
```

```
        this.setState({ name });
      }, 1000);
    }

    generateName = () =>
      this.names[Math.floor(Math.random() * this.names.length)];

    render() {
      return <View name="Sam" />;
    }
  }
```

This component sets up a setInterval in the componentDidMount that sets a randomly picked name from the names array as state. This means that the component gets re-rendered every second because setState is called every second.

The only problem is that our <View /> component gets re-rendered too, even though the name prop is hardcoded (I used my own name there, ‘cause I’m a filthy narcissist).

React.memo to the rescue

memo is a higher order component provided by React that tells the component to only re-render when the props change through the concept of memoization. So let’s wrap our <View /> in a memo!

```
import { memo } from 'React';

const View = memo(({ name }) => `Hi, I'm ${name}`);
```

As you can see in this CodeSandbox example, the <View /> only gets rendered once, because the name prop is hardcoded and doesn’t change. Let’s replace the hardcoded value with the state value (I’ve omitted some lines):

```
class App extends React.Component {
    render() {
      return <View name={this.state.name} />;
    }
  }
```

Questions For Self-Practice / Assignment For The Day

https://au-assignment.s3.ap-south-1.amazonaws.com/Week_21_Day_3_Assignment-1b2cd644-f2b9-47a3-89a4-ce791a424471.pdf